Problem 3: Distributed Coin Flipping Game Using Stellar

Xu Teng (xuteng@iastate.edu)

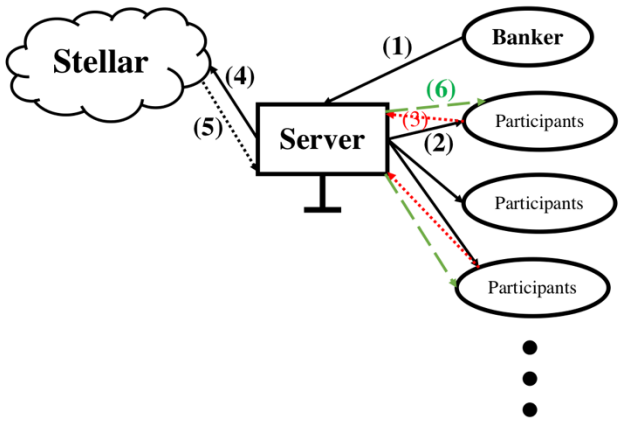
Objective

The objective of this course project is to use Stellar to build a Distributed Online Coin Flipping Game, which allows two or more participants to play the game fairly and transfer funds to each other safely. Stellar is an open platform for building financial products that connect people everywhere. The goal of Stellar is to achieve fast, reliable and near-zero-cost transfers. By using the services provided by the stellar platform, we can build a distributed payment system to support online Coin Flipping game.

Procedure

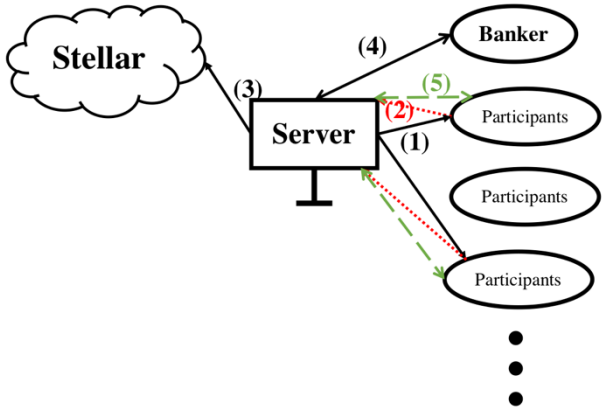
The system consists two user roles: banker and participants, and one server. Whole work is programmed in Java language by Eclipse. Thanks to the experience from second project, CORBA is responsible for dealing with communication (i.e., deliver the requests and return the expected values) between client processes and server process. Since the details of CORBA had already been discussed last time, we would not repeat them again. In the following, the whole gambling procedure is presented.

The right figure shows how a gambling round starts:



(1). Banker sends a message to server with a valid deadline
(2). Server informs all the active participants that a new gambling is on
(3). Participant replies if he/she want to join with a valid bet. At the same time, a random number is generated. Then the bet and the hash value of that number are sent to server
(4). Upon a new message arrival sent from some participant, server records the hash value and applies a transaction from participant to banker in Stellar with the amount of bet
(5). Get the message from Stellar to check if the transaction is successful
(6). Server sends acknowledgement to participant based on the message from (5)

When the current gambling round meeting the deadline:



(1). Server informs the involved participants that the current round ends, and asks participants to enter their random number generated previously
(2). Participants reply with the required information
(3). After receiving all the random numbers from participants and checking if all of them fit the hash value collected previously, server builds a transaction from banker to winner with 95% of the total bets during last round
(4). Banker is notified that the round is over and ask if he/she wants to check the transaction records within one day. If the answer is yes, server will contact Stellar and query the transaction histories within one day
(5). Participants are notified that the game is over and are provided the details of the last round. Then participants are able to ask for the transaction records of previous round

Q: Please explain/prove whether this protocol actually works or not, in term of fairness, and what are the possible attacks (cheating) among the three parties. In your implementation, how have you solved them?

A: With an ideal hash function (i.e., a strict one-way function which is impossible to reveal the unknown original value $\boldsymbol{x}$ from given hash value $\boldsymbol{h(x)}$), this protocol works well since none of the parties is able to hack the random number from others even though gets the hash value of them. But in real world, unfortunately, perfect one-way hash function does not exist, and participant can always infer some information from the hash value. So, to solve this problem, instead of using single hash function, two-round SHA256 is implemented as the hash function to encrypt the original random number twice by SHA256 so as to highly increase the difficulty of hacking/interpreting from hash value to original value.

Measurements

- **Time & Space Complexity**
  All of the involved algorithms are at most linear. And there are four HashMap to store the information of each participants. Therefore, the space complexity is O(# of active users).

- **Latency**
  Since there are two kinds of network communications existing: *Server to Stellar* and *Client to Server*, they are measured individually. Besides, each experiment is repeated 25 times to avoid biased results.
  - *Server to Stellar*: Server sends query to Stellar by interacting with Horizon Java interface. Then Stellar deals with the request and sends the result back to server. The average time cost is 3.246981 s.
  - *Client to Server*: Handled by CORBA. Latency includes network delay and (un)marshalling arguments/results. The average latency is 3.829×10^{-3} s

  Therefore, we can see that the latency caused by CORBA communication is relatively small, while the time cost of Stellar is more than 3 s.

Steps to Compile and Run

1. Run the IDL-to-Java compiler, idlj, to create stubs and skeletons
   `idlj -fall P3.idl`
2. Start orbd and set the port on which the name server to run
   `orbd -ORBInitialPort 1050&`
3. Open Eclipse and load whole project
4. Start Server
   `Find P3Bank.java under “src/(default package)”. Right click on it and choose Run As -> Run Configurations… Under Arguments tab, enter -ORBInitialPort 1050 -ORBInitialHost 127.0.0.1 in Program arguments. Click on Run`
5. Start Banker
   `Find P3Participant.java under “src/(default package)”. Right click on it and choose Run As -> Run Configurations… Under Arguments tab, enter banker -ORBInitialPort 1050 -ORBInitialHost 127.0.0.1 in Program arguments. Click on Run`
6. Start Participants
   `Find P3Participant.java under “src/(default package)”. Right click on it and choose Run As -> Run Configurations… Under Arguments tab, enter participants -ORBInitialPort 1050 -ORBInitialHost 127.0.0.1 in Program arguments. Click on Run`
7. Close the server/client process and kill orbd port.